COMP4434

Individual Project

Prediction Model for Loan Amount

Kim Songi
19085436D

Introduction

The main purpose of machine learning is to generate a model based on real data and predict the output that will occur when other input values are input. This project's main purpose is to predict the loan amount of new users with the use of relevant personal information from AlphaMoney users.

At this point, the most intuitive and simple model we can find is a line. Therefore, the method of analyzing data to find the line that best explains given data is done by the analysis of Linear Regression.

Linear regression can be described as the equation y = mx + b. Since the shape of the line is determined by the slope m and the intercept b, y can be obtained when x is inserted. The goal of the linear regression model is to eventually get the optimum m and b that can minimize the mean of the errors from all data.
There are several methods for learning the model parameters w and b, among which the representative methods are linear regression and ordinary least squares (OLS). Linear regression finds parameters w and b that minimize the mean squared error between the prediction and the target y in the training set.
The mean square error is the squared difference between the predicted value and the target value, added, and then divided by the number of samples.

For a simple linear regression y = m*X + b, it needs only the slope and intercept to predict how the value of Y will change with the value of X, which means there is only one considerable variable.

However, it is not simple enough to explain any case by simple linear regression. This means that there is a lot of variable X to take into consideration. In the train and test set given for the loan amount prediction, multiple number of variables that affect the prediction of the loan amount can be found and this is called multiple linear regression.

Data Processing / Analytics

```
df = pd.read_csv('train.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27000 entries, 0 to 26999
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Customer ID                   27000 non-null  object
 1   Name                          27000 non-null  object
 2   Gender                        26957 non-null  object
 3   Age                           27000 non-null  int64
 4   Income (USD)                  22867 non-null  float64
 5   Income Stability              25494 non-null  object
 6   Profession                    27000 non-null  object
 7   Type of Employment            20456 non-null  object
 8   Location                      27000 non-null  object
 9   Loan Amount Request (USD)     27000 non-null  float64
 10  Current Loan Expenses (USD)   26840 non-null  float64
 11  Expense Type 1                27000 non-null  object
 12  Expense Type 2                27000 non-null  object
 13  Dependents                    24759 non-null  float64
 14  Credit Score                  25473 non-null  float64
 15  No. of Defaults               27000 non-null  int64
 16  Has Active Credit Card        25575 non-null  object
 17  Property ID                   27000 non-null  int64
 18  Property Age                  22621 non-null  float64
 19  Property Type                 27000 non-null  int64
 20  Property Location             26693 non-null  object
 21  Co-Applicant                  27000 non-null  int64
 22  Property Price                27000 non-null  float64
 23  Loan Amount                   26705 non-null  float64
dtypes: float64(8), int64(5), object(11)
memory usage: 4.9+ MB
```

The information of the data frame of the train.csv file shows that there are 24 columns, which means there are 24 different features that can affect the prediction of the loan amount. However, I have decided to drop the Name and Customer columns since they are not powerful features that affect the loan amount.

```
df1 = df.drop('Customer ID', axis=1)
df1 = df1.drop('Name', axis=1)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27000 entries, 0 to 26999
Data columns (total 22 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Gender                        26957 non-null  object
 1   Age                           27000 non-null  int64
 2   Income (USD)                  22867 non-null  float64
 3   Income Stability              25494 non-null  object
 4   Profession                    27000 non-null  object
 5   Type of Employment            20456 non-null  object
 6   Location                      27000 non-null  object
 7   Loan Amount Request (USD)     27000 non-null  float64
 8   Current Loan Expenses (USD)   26840 non-null  float64
 9   Expense Type 1                27000 non-null  object
 10  Expense Type 2                27000 non-null  object
 11  Dependents                    24759 non-null  float64
 12  Credit Score                  25473 non-null  float64
 13  No. of Defaults               27000 non-null  int64
 14  Has Active Credit Card        25575 non-null  object
 15  Property ID                   27000 non-null  int64
 16  Property Age                  22621 non-null  float64
 17  Property Type                 27000 non-null  int64
 18  Property Location             26693 non-null  object
 19  Co-Applicant                  27000 non-null  int64
 20  Property Price                27000 non-null  float64
 21  Loan Amount                   26705 non-null  float64
dtypes: float64(8), int64(5), object(9)
memory usage: 4.5+ MB
```

```
df1.isnull().sum()
```

```
Gender                            43
Age                                0
Income (USD)                    4133
Income Stability                1506
Profession                         0
Type of Employment              6544
Location                           0
Loan Amount Request (USD)          0
Current Loan Expenses (USD)      160
Expense Type 1                     0
Expense Type 2                     0
Dependents                      2241
Credit Score                    1527
No. of Defaults                    0
Has Active Credit Card          1425
Property ID                        0
Property Age                    4379
Property Type                      0
Property Location                307
Co-Applicant                       0
Property Price                     0
Loan Amount                      295
dtype: int64
```

Then I checked the Null value within the data set since the missing values have to be handled before data scaling.

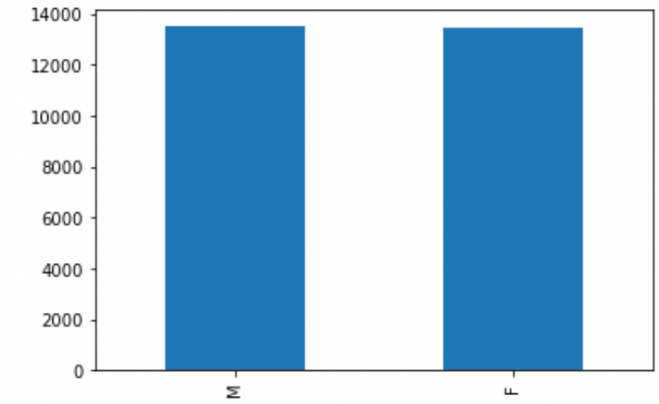
By looking through the train.csv file, I have checked the presence of outliers '-999' in 'Current Loan Expenses (USD)', 'Co-Applicant', 'Property Price', and 'Loan Amount' columns so I have changed the value of outliers into None value first. This process is to remove the presence of outliers and treat them as missing values so that I can preprocess the data.

In [66]:
```
df1.loc[df1['Current Loan Expenses (USD)'] == -999.0, 'Current Loan Expenses (USD)'] = None
df1.loc[df1['Co-Applicant'] == -999, 'Co-Applicant'] = None
df1.loc[df1['Property Price'] == -999.0, 'Property Price'] = None
df1.loc[df1['Loan Amount'] == -999.0, 'Loan Amount'] = None
df1.isnull().sum()
```

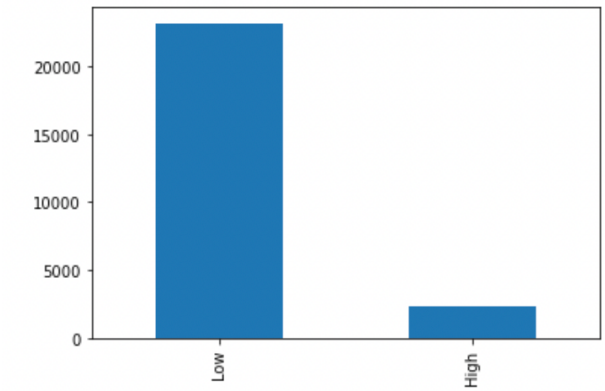
Out[66]:
```
Gender                            43
Age                                0
Income (USD)                    4133
Income Stability                1506
Profession                         0
Type of Employment              6544
Location                           0
Loan Amount Request (USD)          0
Current Loan Expenses (USD)      325
Expense Type 1                     0
Expense Type 2                     0
Dependents                      2241
Credit Score                    1527
No. of Defaults                    0
Has Active Credit Card          1425
Property ID                        0
Property Age                    4379
Property Type                      0
Property Location                307
Co-Applicant                     150
Property Price                   313
Loan Amount                      596
dtype: int64
```

Increased number of null values can be observed from the data frame.

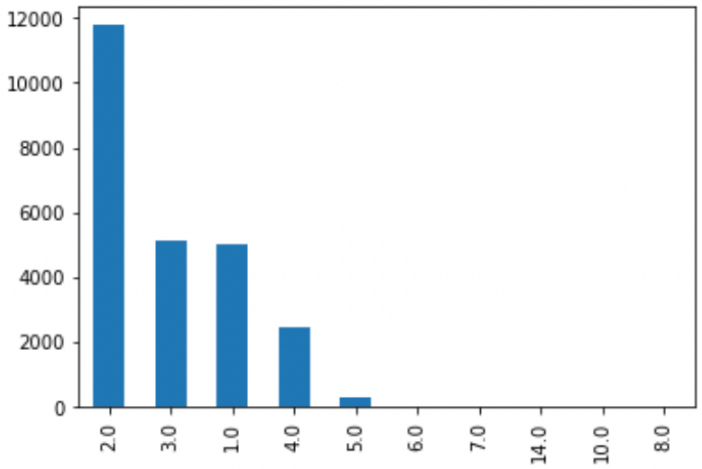
```
df1['Gender'].value_counts().plot(kind='bar')
plt.show()
df1['Income Stability'].value_counts().plot(kind='bar')
plt.show()
df1['Dependents'].value_counts().plot(kind='bar')
plt.show()
df1['Type of Employment'].value_counts().plot(kind='bar')
plt.show()
df1['Has Active Credit Card'].value_counts().plot(kind='bar')
plt.show()
df1['Property Location'].value_counts().plot(kind='bar')
plt.show()
df1['Co-Applicant'].value_counts().plot(kind='bar')
plt.show()
```



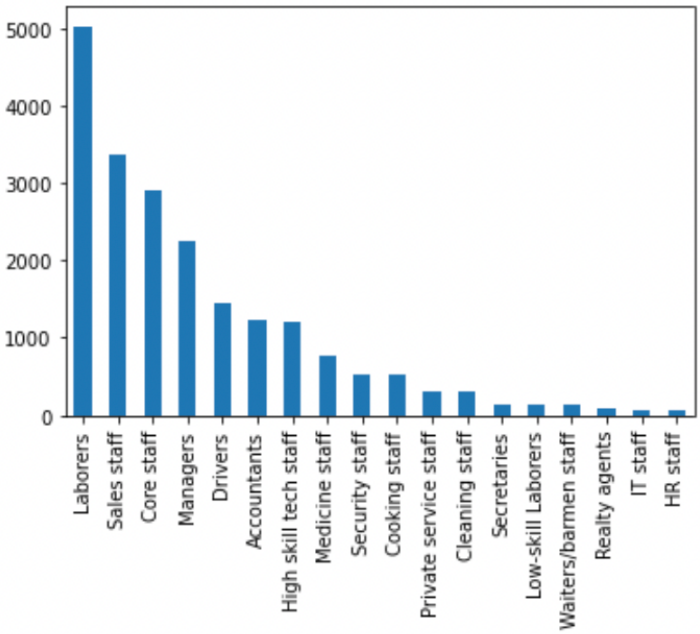
Then, I plotted the bar graphs for each column, which contains missing values, to decide how to fill in the missing values. The above graph is the distribution of gender values, and it shows the almost same distribution.
However, there are only 43 missing values in 27000 data set so I have decided to fill 'M' in the missing fields since missing fields possessed a small portion.



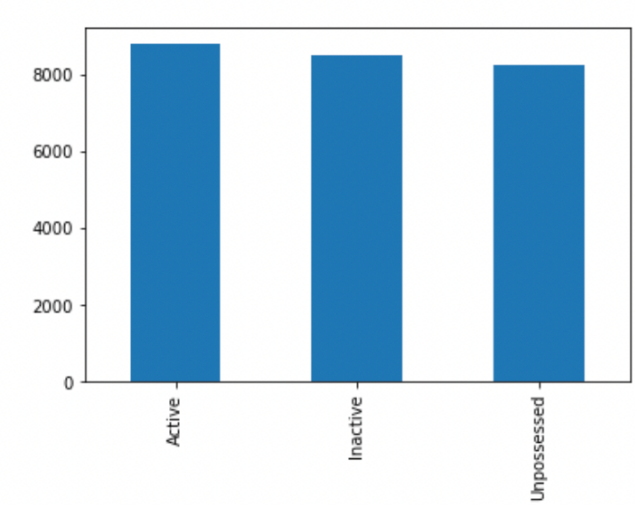
This graph is for the 'Income Stability' column, and it shows the significantly focused distribution to the 'Low' value, so I have decided to replace the null values with 'Low'.



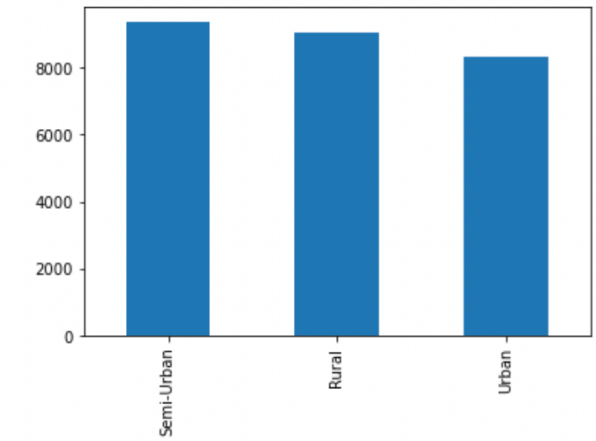
This graph is for the 'Dependents' column and this graph also depicts that the most concentrated value is 2.0 hence, 2.0 is the value that I chose to replace the null values.



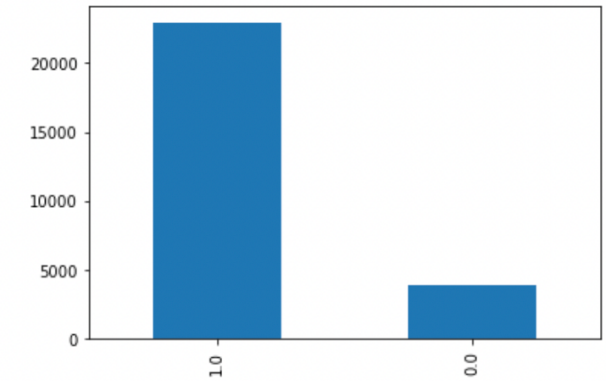
This graph is for the 'Type of Employment' column and the most concentrated value is Laborers.



The above graph is for the ‘Has Active Credit Card’ column and it shows a similar concentration in the distribution of values. However, the most frequently shown value is ‘Active’ hence, it is chosen as the replacement.



The graph for the ‘Property Location’ column also shows the fair distribution of values. However, I used the most frequently shown value ‘Semi-Urban’ as the replacing value.



The graph for the ‘Co-Applicant’ column shows the concentrated distribution on value ‘1.0’.

```
df1['Gender'] = df1['Gender'].fillna('M')
df1['Income (USD)'] = df1['Income (USD)'].fillna(df1['Income (USD)'].mean())
df1['Income Stability'] = df1['Income Stability'].fillna('Low')
df1['Type of Employment'] = df1['Type of Employment'].fillna('Laborers')
df1['Dependents'] = df1['Dependents'].fillna(2.0)
df1['Credit Score'] = df1['Credit Score'].fillna(df1['Credit Score'].mean())
df1['Property Age'] = df1['Property Age'].fillna(df1['Property Age'].mean())
df1['Co-Applicant'] = df1['Co-Applicant'].fillna(1)
df1['Property Price'] = df1['Property Price'].fillna(df1['Property Price'].mean())
df1['Current Loan Expenses (USD)'] = df1['Current Loan Expenses (USD)'].fillna(df1['Current Loan Expenses (USD)'].mean(
df1['Has Active Credit Card'] = df1['Has Active Credit Card'].fillna('Active')
df1['Property Location'] = df1['Property Location'].fillna('Semi-Urban')
df1.isnull().sum()
```

```
Gender                            0
Age                               0
Income (USD)                      0
Income Stability                  0
Profession                        0
Type of Employment                0
Location                          0
Loan Amount Request (USD)         0
Current Loan Expenses (USD)       0
Expense Type 1                    0
Expense Type 2                    0
Dependents                        0
Credit Score                      0
No. of Defaults                   0
Has Active Credit Card            0
Property ID                       0
Property Age                      0
Property Type                     0
Property Location                 0
Co-Applicant                      0
Property Price                    0
Loan Amount                     596
dtype: int64
```

The null values in the rest of the columns mostly with number values were replaced with the mean value of each column.
Hence, based on the data distribution and mean value, the replacement of null values was done and checked through df1.isnull().sum() command.

```
df2 = df1.dropna(axis=0)[['Gender','Age', 'Income (USD)', 'Income Stability', 'Profession','Type of Employment',
                          'Location', 'Loan Amount Request (USD)', 'Current Loan Expenses (USD)', 'Expense Type 1',
                          'Expense Type 2', 'Dependents', 'Credit Score', 'No. of Defaults', 'Has Active Credit Card',
                          'Property ID','Property Age', 'Property Type', 'Property Location', 'Co-Applicant',
                          'Property Price', 'Loan Amount']]
print(df2.shape)
df2.isnull().sum()
```

```
(26404, 22)
```

```
Gender                            0
Age                               0
Income (USD)                      0
Income Stability                  0
Profession                        0
Type of Employment                0
Location                          0
Loan Amount Request (USD)         0
Current Loan Expenses (USD)       0
Expense Type 1                    0
Expense Type 2                    0
Dependents                        0
Credit Score                      0
No. of Defaults                   0
Has Active Credit Card            0
Property ID                       0
Property Age                      0
Property Type                     0
Property Location                 0
Co-Applicant                      0
Property Price                    0
Loan Amount                       0
dtype: int64
```

For the null values in the ‘Loan Amount’ column, I have decided to drop the corresponding rows since the model is to predict the loan amount, values of loan amount should be the most accurate.

```
x_train = df2[['Gender','Age', 'Income (USD)', 'Income Stability', 'Profession', 'Type of Employment', 'Location',
                          'Loan Amount Request (USD)', 'Current Loan Expenses (USD)', 'Expense Type 1',
                          'Expense Type 2', 'Dependents', 'Credit Score', 'No. of Defaults', 'Has Active Credit Card',
                          'Property ID','Property Age', 'Property Type', 'Property Location', 'Co-Applicant',
                          'Property Price']]
y_train = df2[['Loan Amount']]
x_train.shape, y_train.shape
```

```
((26404, 21), (26404, 1))
```

After checking that there are no missing values in the data frame, the data is now divided into x and y values. x_train stores the features affecting the y value and y_train stores the 'Loan Amount' values.

Since the machine learning algorithm provided by scikit-learn does not accept string values as input values, it is necessary to train the machine learning model after preprocessing to encode all string values into numeric types. Hence, the data value of a column that is a string needs to be changed to a numeric data value. This can be done by One-hot encoding and to express the relationship between data independently, one-hot encoding creates as many columns as the number of unique values, then put True in one column and False in the rest. And there is a handy one-hot encoding function that can be done with Pandas, which is the pd.get_dummies function.

```
cols = x_train.select_dtypes('object').columns.values
cols
x_train1 = pd.get_dummies(data=x_train, columns=cols)
x_train1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26404 entries, 0 to 26999
Data columns (total 55 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Age                                   26404 non-null  int64
 1   Income (USD)                          26404 non-null  float64
 2   Loan Amount Request (USD)             26404 non-null  float64
 3   Current Loan Expenses (USD)           26404 non-null  float64
 4   Dependents                            26404 non-null  float64
 5   Credit Score                          26404 non-null  float64
 6   No. of Defaults                       26404 non-null  int64
 7   Property ID                           26404 non-null  int64
 8   Property Age                          26404 non-null  float64
 9   Property Type                         26404 non-null  int64
 10  Co-Applicant                          26404 non-null  float64
 11  Property Price                        26404 non-null  float64
 12  Gender_F                              26404 non-null  uint8
 13  Gender_M                              26404 non-null  uint8
 14  Income Stability_High                 26404 non-null  uint8
 15  Income Stability_Low                  26404 non-null  uint8
 16  Profession_Businessman                26404 non-null  uint8
 17  Profession_Commercial associate       26404 non-null  uint8
 18  Profession_Maternity leave            26404 non-null  uint8
 19  Profession_Pensioner                  26404 non-null  uint8
 20  Profession_State servant              26404 non-null  uint8
 21  Profession_Student                    26404 non-null  uint8
 22  Profession_Unemployed                 26404 non-null  uint8
 23  Profession_Working                    26404 non-null  uint8
```

```
 24  Type of Employment_Accountants            26404 non-null  uint8
 25  Type of Employment_Cleaning staff         26404 non-null  uint8
 26  Type of Employment_Cooking staff          26404 non-null  uint8
 27  Type of Employment_Core staff             26404 non-null  uint8
 28  Type of Employment_Drivers                26404 non-null  uint8
 29  Type of Employment_HR staff               26404 non-null  uint8
 30  Type of Employment_High skill tech staff  26404 non-null  uint8
 31  Type of Employment_IT staff               26404 non-null  uint8
 32  Type of Employment_Laborers               26404 non-null  uint8
 33  Type of Employment_Low-skill Laborers     26404 non-null  uint8
 34  Type of Employment_Managers               26404 non-null  uint8
 35  Type of Employment_Medicine staff         26404 non-null  uint8
 36  Type of Employment_Private service staff  26404 non-null  uint8
 37  Type of Employment_Realty agents          26404 non-null  uint8
 38  Type of Employment_Sales staff            26404 non-null  uint8
 39  Type of Employment_Secretaries            26404 non-null  uint8
 40  Type of Employment_Security staff         26404 non-null  uint8
 41  Type of Employment_Waiters/barmen staff   26404 non-null  uint8
 42  Location_Rural                            26404 non-null  uint8
 43  Location_Semi-Urban                       26404 non-null  uint8
 44  Location_Urban                            26404 non-null  uint8
 45  Expense Type 1_N                          26404 non-null  uint8
 46  Expense Type 1_Y                          26404 non-null  uint8
 47  Expense Type 2_N                          26404 non-null  uint8
 48  Expense Type 2_Y                          26404 non-null  uint8
 49  Has Active Credit Card_Active             26404 non-null  uint8
 50  Has Active Credit Card_Inactive           26404 non-null  uint8
 51  Has Active Credit Card_Unpossessed        26404 non-null  uint8
 52  Property Location_Rural                   26404 non-null  uint8
 53  Property Location_Semi-Urban              26404 non-null  uint8
 54  Property Location_Urban                   26404 non-null  uint8
dtypes: float64(8), int64(4), uint8(43)
memory usage: 3.7 MB
```

After applying the pd.get_dummies function to every column with data type ‘object’, an increased number of columns can be observed with data type uint8.
Then the data is ready to be scaled.
Before modelling data, it must go through a scaling process. Scaling makes it easy to compare and analyze multidimensional values, prevents data overflow or underflow, and reduces the condition number of the covariance matrix of independent variables in the optimization process. Improves stability and convergence speed.

I have chosen the MinMaxScaler as my scaler.
MinMaxSacler is the scaling mean that scales to make max/min values 1 and 0 respectively. Rescale the data so that all feature values are between 0 and 1. However, if there is an outlier, the transformed value may be compressed into a very narrow range.

```
scaler = MinMaxScaler()
x_train1 = scaler.fit_transform(x_train1)
```

I have applied the same method to the test.csv data for data preprocessing – removing outliers, replacing null values, one-hot encoding, and data scaling.

Model design and Implementation

I have used sklearn.linear_model.LinearRegression() class for the linear regression model implementation. This is an ordinary least squares linear regression, and it fits a linear model with coefficients w = (w1, …, wp) that minimizes the RSS (residual sum of squares) between the observed target data and the target predicted by a linear relationship.

```
model = LinearRegression()
model.fit(x_train1, y_train)
```

The fit method is used to train the data using x_train1 and y_train datasets. The x_train1 data is used as training data and y_train data is the target values.

```
x_test = scaler.fit_transform(x_test)
y_pred = model.predict(x_test)
```

The above screenshot shows the step for scaling the data and putting the test dataset into the trained linear model to estimate the loan amount and this is done by using the predict method.

Performance evaluation and discussions

Linear Regression model

```
model.score(x_train1,y_train)
```

```
0.6597095293647716
```

I have used the score function to evaluate the accuracy of the trained model.
The score function returns a coefficient of determination called R^2 and the larger the coefficient of determination R^2, the more similar the actual and predicted values are, which means the better the data is explained.

The value I got for the score is 0.659709… and it can be rounded up to 0.66. This value represents that 66% of the loan amount in the y_train dataset can be explained with input features in x_train1.

However, when I tried training the model after dropping several columns, I could observe that the R^2 value is smaller than the above. The above value is achieved when I trained using every column, except name and customer ID, which can be seen as overfitting of the data that might lead to a less accurate model than the other data set.

Summary and Future work

In one-dimensional datasets, the model can be very simple, so there is no need to worry about overfitting. However, for high-dimensional datasets with many features, the performance of the linear model is very high, and there is a possibility of overfitting.

Since overfitting is the possible problem in the linear regression model of this project, the other regression method such as Ridge can be used to control the complexity of the model and then resolve the overfitting problem. This may reduce the accuracy of the train set but it results in a more generalized model.

Reference

Stojiljković, M. (2021, March 19). Linear regression in python. Retrieved April 10, 2022, from https://realpython.com/linear-regression-in-python/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

Brownlee, J. (2020, August 27). How to use StandardScaler and MinMaxScaler transforms in Python. Retrieved April 10, 2022, from https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/